AN APPROACH TO THE INCREASE OF THE PERFORMANCE OF VLSI SYSTEM BY IMPLEMENTING SYSTOLIC ARRAY

楊　立　人

（作者爲本校應用數學系兼任講師）

摘　　要

本文主要是研究有關電腦中微處理機內部資料傳送之另一簡捷方法。當初美國卡內基——梅隆大學（Carnegic-Mellon University）電腦系孔教授（Prof. H. T. KUNG）之所以取收縮陣列來做此種傳送方法之名是因爲此種處理方法有如人體心臟之跳動，十分有節奏一放一縮的運作著。

有關收縮陣列方面之論著已不在少數，但直至目前爲止，彼等之解決方法大致只能將（n+1）×（n+1）之線性方程式系統所用之傳輸時間減至 Bareiss 所提出之 Order（n^2）或 Bitmead & Anderson 所提出的 Order（$n\log^2 n$）。如果n 値不大，Order（$n\log^2 n$）所花的時間會較長，換句話說，即其速度反而會比 Order（n^2）要來得慢，同時其解法也比較繁複。

據本文以 Bareiss 所提出之解法做修正並推衍之結果，吾人理論上似可利用一維之收縮陣列將超大型電腦之線性方程式系統使用之時間與記憶空間具減爲 Order（n）。

ABSTRACT

A systolic system is a network of processors which rhythmically compute and pass data through the system. Many basic matrix computations can be pipelined efficiently on systolic networks having an array structure. In this paper we are trying to find a better solution for an (n+1) x (n+1) system of linear equations by using a one-dimensional systolic array. In the algorithms implemented shown that it requires only order(n) time and order(n) storage.

1. Introduction

Systolic array has been used so widely during the recent years in solving different problems occured or being found in the VLSI system. Charles E. Leiserson proposed a systolic priority queues in the VLSI circuits, H.T. Kung has designed algorithms for VLSI chips. Recently Bitmead and Anderson proposed procedures which require only order $(n\log^2 n)$ time and order(n) space when applied to order(n+1) system. We found that these procedures are kind of too complicated and if n is not really large then its speed would be even slower than the order(n^2)-time methods which has been proposed by Bareiss. In order to find a better solution which can compete with the previous researches, we have implemented and modified Bareiss algorithms which have been proved requires only O(n) processors. The results we have got are quite excited. After the new implementations, we have reduced the storage requirements from $O(n^2)$ down to $O(n)$.

2. Brief Review of the Bariess Algorithm

At the beginning, let us review some of the algorithms developed by Bariess. Suppose there is a matrix of order(n+1) and, we named it as T which respond to a column vector B. In short, it is Tx = B,
and after extension, it becomes

$$\begin{bmatrix} t_0 & t_1 & t_2 & - & - & - & - & - & - & - & t_n \\ t_{-1} & t_0 & t_1 & - & - & - & - & - & - & - & t_{n-1} \\ t_{-2} & t_{-1} & t_0 & - & - & - & - & - & - & - & t_{n-2} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ t_{-n} & t_{-n+1} & t_{-n+2} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & t_0 \end{bmatrix} x = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_n \end{bmatrix}$$

Now, let us take a look at Bariess two examples listed in p. 415 [3].

Example 1.

$$T = 120 \quad \begin{matrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 \\ 4 & 3 & 2 & 1 & 2 \\ 5 & 4 & 3 & 2 & 1 \end{matrix}$$

4.3		Shift ←				120				
4.2 m_3	=	$\frac{24}{-288} = -\frac{1}{12}$				120	0			
3.3		Shift ←			144	120	24			
3.2 m_2	=	$\frac{30}{-300} = -\frac{1}{10}$			144	120	0	24		
2.3		Shift ←		180	150	120	30	60		
2.2 m_2	=	$\frac{40}{-320} = -\frac{1}{8}$		180	150	120	0	30	60	
1.3		Shift ←	240	200	160	120	40	80	120	
1.2 m_1	=	$\frac{240}{-360} = -\frac{2}{3}$	240	200	160	120	0	40	80	120
			480	360	240	120	240	360	480	600
			600	480	360	240	120	240	360	480
1.1 m_{-1}	=	$\frac{240}{120} = 2$	-360	-240	-120	0	-360	-480	-600	-720
1.3		Shift →		-360	-240	-120	-360	-480	-600	-720
2.1 m_{-2}	=	$\frac{-120}{120} = -1$		-160	-80	0	-320	-400	-480	
2.3		Shift →			-160	-80	-320	-400	-480	
3.1 m_{-3}	=	$\frac{-80}{120} = -\frac{2}{3}$			-60	0	-300	-360		
3.3		Shift →				-60	-300	-360		
4.1 m_{-4}	=	$\frac{-60}{120} = -\frac{1}{2}$				0	-288			
4.3		Shift →					-288			
			p_{-3}	p_{-2}	p_{-1}		p_0	p_1	p_2	p_3

Hence

$$T^{(-4)} = \begin{array}{ccccc} 120 & 240 & 360 & 480 & 600 \\ & -360 & -480 & -600 & -720 \\ & & -320 & -400 & -480 \\ & & & -300 & -360 \\ & 0 & & & -288 \end{array}$$

and

$$T^{(4)} = \begin{matrix} 120 & & & 0 & \\ 144 & 120 & & & \\ 180 & 150 & 120 & & \\ 240 & 200 & 160 & 120 & \\ 600 & 480 & 360 & 240 & 120 \end{matrix}$$

Example 2.

$$b = 120 \begin{matrix} 30 \\ 22 \\ 18 \\ 20 \\ 30 \end{matrix}$$

4.2 $m_4 = -\frac{1}{12}$		120			
3.2 $m_3 = -\frac{1}{10}$		120	384		
2.2 $m_2 = -\frac{1}{8}$		240	390	840	
1.2 $m_1 = -\frac{2}{3}$		560	560	880	1600
		3600	2640	2160	2400
		2640	2160	2400	3600
1.1 $m_{-1} = 2$		-4560	-3120	-1920	-1200
1.3 Shift ←	-4560	-3120	-1920	-1200	
2.1 $m_{-2} = -1$		-2560	-1360	-320	
2.3 Shift ←	-2560	-1360	-320		
3.1 $m_{-3} = -\frac{2}{3}$		-1200	-60		
3.3 Shift ←	-1200	-60			
4.1 $m_{-4} = -\frac{1}{2}$		0			
4.3 Shift ←	0				
		Q_0	Q_1	Q_2	Q_3

$$b^{(-4)} = \begin{matrix} 3600 \\ -4560 \\ -2560 \\ -1200 \\ 0 \end{matrix}$$

According to Bareiss algorithm $Tx = B$ can be transformed into $T^{(-1)}x = B^{(-1)}$, $T^{(1)}x = B^{(1)}$, $T^{(-2)}x = B^{(-2)}$, $T^{(2)}x = B^{(2)}$, ... $T^{(-n)}x = B^{(-n)}$, $T^{(n)}x = B^{(n)}$
and the $T^{(-n)}$; $T^{(n)}$ are upper and lower triangulars respectively.

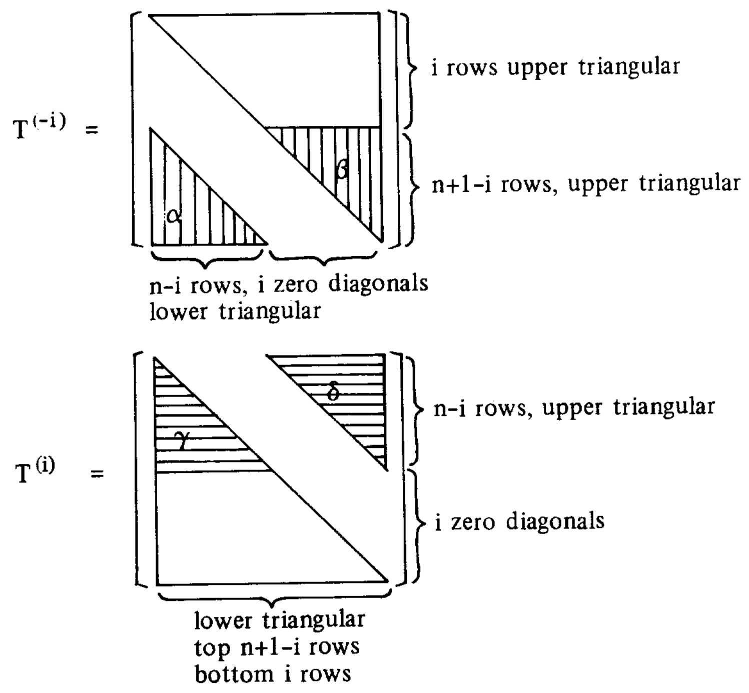

Figure 1. Structure of $T^{(-i)}$ and $T^{(i)}$ in the Bareiss algorithm

We introduce the shift matrices

$$Z_{-i} = (Z_{jk}^{(-i)}) = \delta_{j-k-i}$$

$$Z_i = (Z_{jk}^{(i)}) = \delta_{j-k+i}$$

The effect of premultiplication by Z_{-i} is to downshift the matrix $T^{(i-1)}$ by i rows and to replace its first i rows by zeros. Similarly, Z_i upshifts $T^{(-i)}$ by i rows and replaces its last i rows by zeros. Suppose that the matrix $T^{(-i+1)}$ has (i-1) null subdiagonals. The following equations show that it will wipe the i-th subdiagonal out without disturbing those already null elements. It follows that the matrix $T^{(-n)}$ will be upper triangular.

$$T^{(-i)} = T^{(-i+1)} - m_{-i}Z_{-i}T^{(i-1)}, \text{ with } m_{-i} = \frac{t_{i,0}^{(1-i)}}{t_0}$$

$$T^{(i)} = T^{(i-1)} - m_{-i}Z_iT^{(-i)}, \text{ with } m_i = \frac{t_{0,1}^{(i-1)}}{t_{n,n}^{(-i)}}$$

$$B^{(-i)} = B^{(-i+1)} - m_{-i}Z_{-i}B^{(i-1)}$$

$$B^{(i)} = B^{(i-1)} - m_i Z_i B^{(-i)}$$

The Bareiss method will not fail as long as all the leading principal submatrices of the given matrix T are nonsingular. An LU-factorization, with L unit lower triangular, of T is then given.

Where $T = LU$

$$L = \frac{1}{t_0}(T^{(n)})^{T2}$$

$$U = T^{(-n)}$$

3. New Approach for the Solution

We present here a one-dimensional systolic array for computing the two triangular matices $T^{(-n)}$ and $T^{(n)}$. The array consists of $2n-1$ processors P_{-n+1}, P_{-n+2}, ... P_{-1}, P_0, P_1, . . . , P_{n-1}, arranged from left to right. All processors are identical except for the middle one P_0. For simplicity we first assume that P_0 can broadcast a scalar quantity to all other processors in constant time. This assumption will soon be dropped. The processor P_0 has four registers U_{0-}, U_{0+}, D_{0-}, and D_{0+}, and each remaining processor P_k (k not equals to 0) has two registers U_k and D_k. We denote the content of register R by [R]. Each processor has two output lines $outu_k$ and $outd_k$, and two input lines inu_k and ind_k. The output line $outu_k$ is connected to the input line inu_{k-1}, for $k = 1, 2, \ldots n-1$. The output line $outd_{-k}$ is connected to the input line ind_{-k+1}, for $k = 1, 2, \ldots . n-1$.

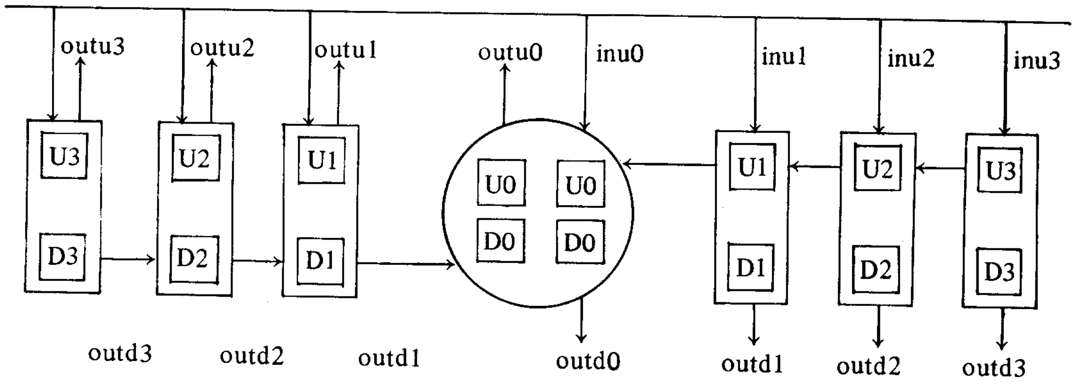

Figure 2. The processor array for n = 4

Before iterating, we feed data into the array so that

$$[U_{0-}] = t_0 \qquad [U_{0+}] = t_1,$$

$$[D_{0-}] = t_{-1} \qquad [D_{0+}] = t_0$$

and

$$[U_{-k}] = t_{-k}, \qquad [U_k] = t_{k+1},$$

$$[D_{-k}] = t_{-k-1} \qquad [D_k] = t_k$$

for $k=1,2,\ldots,n-1$, we now consider one iteration, say the i-th one. Each iteration consists of three steps. At step one the processor P_0 computes the multiplier

$$m_{-i} = \frac{[D_{0-}]}{[U_{0-}]}$$

This value is broadcast to all processors. Processor P_k, all k, now computes

$$(i) \equiv [D_k] - m_{-i}[U_k]$$

The quantity D_k, for $k \leq 0$ is output on line out_k, and the register U_k receives the value of (ii). At the third st :p, we do data transfer. The content of register D_k, for k-1, is sent to processor P_{k+1} on line $outd_k$. The incoming data for P_{k+1} is stored in register D_{k+1}. The content of register D_{0-} is lost. At the same time, the content of register U_k, for $k \geqslant 1$ is sent to processor P_{k-1}, on line $outu_k$. The incoming data for P_{k-1} is stored in register U_{k-1}, and the content of register U_{0+} is lost. We now disable processors P_{-n+i} and P_{n-i}. This completes one iteration of the Bareiss algorithm.

Since we disable the two end processors after each iteration, our method must terminate after n iterations. Let us denote the output on line $outd_k$ at the i-th iteration by $d_k^{(i)}$ and the corresponding output on $outu_k$ by $u_k^{(i)}$. The two desired matrices $T^{(-n)}$ and $T^{(n)}$ are given by

$$T^{(-n)} = \begin{array}{cccccccc} t_0 & t_1 & t_2 & t_3 & \cdot & \cdot & \cdot & t_n \\ & d_0^{(1)} & d_1^{(1)} & d_2^{(1)} \cdot & \cdot & & \cdot & d_{n-1}^{(1)} \\ & & d_0^{(2)} & d_1^{(2)} \cdot & \cdot & & \cdot & d_{n-2}^{(2)} \\ & & & d_0^{(3)} \cdot & \cdot & & \cdot & d_{n-3}^{(3)} \\ & & & & \cdot & & \cdot & \\ & & & & & & \cdot & \\ 0 & & & & & & & d_0^{(n)} \end{array} \qquad \text{and}$$

$$T^{(n)} = \begin{array}{ccccccc}
u_0^{(n)} & & & & & & \\
u_{-1}^{(n-1)} & u_0^{(n-1)} & & & & & \\
u_{-2}^{(n-2)} & u_{-1}^{(n-2)} & u_0^{(n-2)} & & & & \\
\cdot & \cdot & \cdot & \cdot & & & \\
\cdot & \cdot & \cdot & \cdot & \cdot & & \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \\
u_{-n+1}^{(1)} & u_{-n+2}^{(1)} & u_{-n+3}^{(1)} & \cdot \quad \cdot & \cdot \quad \cdot & u_0^{(1)} & \\
t_{-n} & t_{-n+1} & t_{-n+2} & & & t_1 & t_0
\end{array}$$

By transforming the right-hand vector B simultaneously (we will discuss it later). We obtain an upper triangular system $T^{(-n)}x = B^{(-n)}$ to solve. So $T^{(n)}$ can be discarded. The Bareiss algorithm appears to require order (n^2) storage. However, at the expense of some extra computation, we can avoid using more than order(n) storage (the details are presented in the later part of this section too).

We show now that broadcasting is not needed if each processor P_k (resp. P_{-k}), $k > 0$, can pass a scalar quantity to its outer neighbor P_{k+1} (resp. P_{-k-1}) and if P_0 can pass a number to both P_1 and P_{-1}. Let us describe how the i-th iteration (say) proceeds. The middle processor P_0 reads the inputs on lines inu_0 and ind_0, and stores the numbers in registers U_{0+} and D_{0-}, respectively. The computing starts with P_0 calculating the multiplier m_{-i} and passes the value to its two neighbors P_{-1} and P_1. The processor P_0 now performs the second step of the iteration by computing the multiplier m_i and again passes its value to processors P_{-1} and P_1. The processor P_1 (resp. P_{-1}), on receiving m_{-i}, will do the computation and pass the multiplier to the neighbor P_2 (resp. P_{-2}). Then P_1 (resp. P_{-1}) will receive the multiplier m_i. The operations are now performed and the value of m_i will be passed to processor P_2 (resp. P_{-2}). The processor P_1 (resp. P_{-1}) completes its share of the iteration by sending the content of register U_1 (resp. D_{-1}) leftward (resp. rightward) to processor P_0. The processors P_2 and P_{-2} on receiving the multipliers m_{-i} and then m_i, will perform the necessary computations, the passing on of the multipliers and finally the shifting of the necessary information toward the middle processor.

This process expands outward until the two end processors P_{n-i} and P_{-n+i} have done their tasks, ending the iteration. An important observation here is that the (i+1)-st iteration can start as soon as processor P_0 receives the data from its two neighbors. Our technique for avoiding broadcast is quite common: processors are active only on alternate time steps (P_0, P_{+2} .... at time $\tau = 1,3....$ and $P_{\pm 1}, P_{\pm 3}, ...$

at time $\tau = 2,4\ldots$), and the operations of processors $P_{\pm k}$ are delayed by k time steps relative to the operation of processor P_0.

Suppose that the given matrix T is banded with half-bandwidth w. We can, of course, disregard this special structure and still use 2n-1 processors. But as processors $P_{\pm(w-1)}, P_{\pm w}, \ldots P_{\pm(n-1)}$ work only with null data, we may perform the elimination using only 2w-3 processors if the input lines inu_{w-2} and ind_{-w+2} always carry the number zero. We will disable the processors P_{n-i} and P_{-n+i} at the end of the i-th iteration, for i = n-w+2, n-w+3, ... n-1.

Now let us look back to Bareiss example 1.

Note that T = LU, where $U = T^{(-4)}$ and

$$L = \frac{1}{120}(T^{(4)})^{T2} = \begin{array}{ccccc} 1 & & & & 0 \\ 2 & 1 & & & \\ 3 & \frac{4}{3} & 1 & & \\ 4 & \frac{5}{3} & \frac{5}{4} & 1 & \\ 5 & 2 & \frac{3}{2} & \frac{6}{5} & 1 \end{array}$$

Suppose that we want to find the vector $B^{(-n)}$ that satisfies $T^{(-n)}x = B^{(-n)}$ then we have to do the following steps that is what we have mentioned before, modifying the right-hand vector.

The structure and the operations of the systolic array are very similar to the right-half of the systolic architecture of the previous section. The array here consists of the n processors $Q_0, Q_1, \ldots, Q_{n-1}$. All the processors are identical.

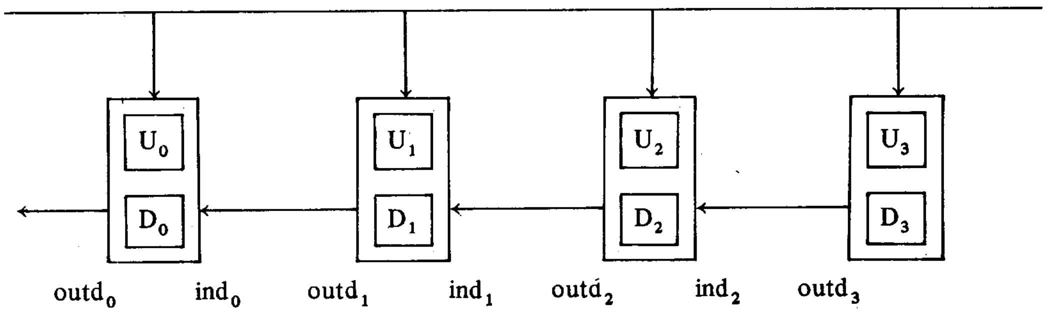

Figure 3. Systolic array for finding $B^{(-n)}$ (n = 4)

Each Q_k has two registers U_k and D_k, one output line $outd_k$ and one input line ind_k. The lines ind_k and $outd_{k+1}$ are connected (for $k = 0, 1, \ldots, n-2$). We also assume temporarily that all processors can receive the broadcast of a scalar quantity.

Before iteration starts, data are fed into the processors so that, for $k = 0, 1, \ldots, n-1$,

$$[U_k] = B_k$$

$$[D_k] = B_{k+1}$$

where $B = (b_0, b_1, \ldots, b_n)^T$. Let us consider the i-th iteration, where $i \geqslant 1$. An iteration consists of three steps, same as in the previous section. At the first step, the multiplier m_{-i} arrives at processor Q_k, which then computes

$$\text{(iii)} \equiv [D_k] - m_{-i}[U_k]$$

and stores (iii) in register D_k. The second step begins when Q_k receives the multiplier m_i. It computes

$$\text{(iv)} \equiv [U_k] - m_i[D_k]$$

and stores the result in U_k. The third step is now initiated. It involves a transfer of the content of register D_k from processor Q_k to processor Q_{k-1}, for $k \geqslant 1$. The content of D_1 is sent out on $outd_0$ and register D_k, for $k \geqslant 0$, receives the content of register D_{k+1}. The completion of the data transfer ends the i-th iteration and processor Q_{n-i} is disabled. If we denote the output on line $outd_0$ after the i-th iteration by $d_0^{(k)}$, then the vector $B^{(-n)}$ is given by

$$B^{(-n)} = \begin{matrix} b \\ d_0^{(1)} \\ d_0^{(2)} \\ \cdot \\ \cdot \\ \cdot \\ d_0^{(n)} \end{matrix}$$

Since the algorithms in this and the previous sections are very similar, we can argue using the same reasonings as before that broadcasting is unnecessary if each processor Q_k can pass the multiplier to its right neighbor Q_{k+1} and if the operation of processor Q_k is delayed by k time steps relative to the operation of processor Q_0.

And now let us refer back to Bareiss example 2.

We consider the regeneration of the upper triangular matrix $T^{(-n)}$ using only its last column and the 2n multipliers $m_{\pm i}$. Our key idea is to run the elimination algorithm in the previous part of this section:

$$T^{(i-1)} = T^{(i)} + m_i Z_i T^{(-i)},$$

$$T^{(-i+1)} = T^{(-i)} + m_{-i} Z_{-i} T^{(i-1)},$$

for $i = n, n-1, \ldots, 1$. (Observe that rows 0 to i for $T^{(-n)}$ are equal to rows 0 to i of $T^{(-i)}$). So our systolic array consists of n identical processors $B_0, B_1, \ldots, B_{n-1}$. Each processor B_k has two registers U_k and D_k. Initially.

$$[U_k] = 0, \text{ and}$$

$$[D_k] = t^{(-n)}_{n-k,n},$$

for $k = 0, 1, \ldots, n-1$. We again assume for a moment that there is a broadcasting mechanism. Each processor B_k has two output lines $outu_k$ and $outd_k$ and one input line inu_k. The lines $outu_k$ and inu_{k+1} are connected, for $k \geq 0$.

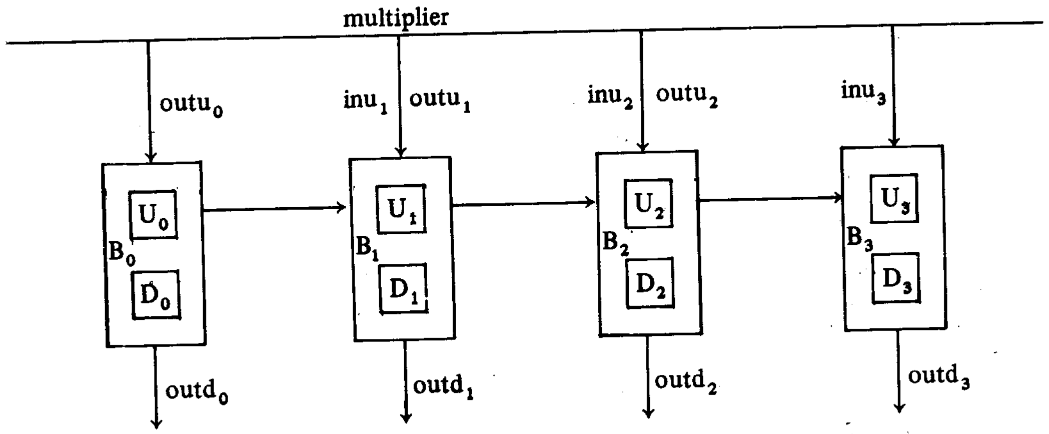

Figure 4. A systolic array for generating $T^{(-n)}$ (n = 4)

Only one processor, B_0, is active for the initial iteration. At the end of the i-th iteration, $i \geq 1$, processor B_i is activated for subsequent computations so that the (i+1) processors $B_0, B_1, \ldots, B_i$ are active during the (i+1)-st iteration. Each iteration consists of three steps. Let us describe the i-th iteration. At the first step, the multiplier m_{n+1-i} is broadcast to all the processors and the following computation is done:

$$(v) \equiv [U_k] + m_{n+1-i}[D_k],$$

for $k=0,1,\ldots,i-1$. The result (v) is stored in register U_k. The second step starts when the multiplier m_{-n-1+i} is broadcast to all processors. Processor B_k $(0 \leqslant k \leqslant i-1)$ then computes

$$(vi) \equiv [D_k] + m_{-n-1+i}[U_k],$$

outputs the result (vi) on outd$_k$ and also stores the number in register D_k. The third step is put a shifting of the content of register U_k to register U_{k+1}, for $k=0,1,\ldots,$ i-1. Register U_0 will contain the number zero. The complete procedure stops after n iterations.

If we denote the output on line outd$_k$ at the i-th iteration by $d_k^{(i)}$, the desired matrix $T^{(-n)}$ is given by

$$T^{(-n)} = \begin{array}{cccccc} d_0^{(n)} & d_1^{(n)} & \cdots & & d_{n-1}^{(n)} & t_{0,n}^{(-n)} \\ & d_0^{(n-1)} & \cdots & & d_{n-2}^{(n-1)} & t_{1,n}^{(-n)} \\ & & \cdots & & \cdot & \cdot \\ & & & \cdot\ \cdot & \cdot & \cdot \\ & & & d_0^{(2)} & d_1^{(2)} & t_{n-2,n}^{(-n)} \\ 0 & & & & d_0^{(1)} & t_{n-1,n}^{(-n)} \\ & & & & & t_{n,n}^{(-n)} \end{array}$$

As before, we can argue that broadcasting is unnecessary as long as each processor can pass a scalar quantity to its right neighbor.

Since our primary concern is the solution of $T^{(-n)}x = B^{(-n)}$ on a linear systolic array, it is interesting to note that we have regenerated the elements of $T^{(-n)}$ in the exact order as required by the Kung-Leiserson algorithm for back substitution.

4.3 shift →	0	240	360	480
4.1 $m_1 = -\frac{2}{3}$	240	350	480	600
3.3 shift →	0	40	80	120
3.1 $m_2 = -\frac{1}{8}$	40	80	120	
2.3 shift →	0	30	60	

2.1 $m_3 = -\frac{1}{10}$	30	60		
1.3 shift →	0	24		
1.1 $m_4 = -\frac{1}{12}$	24			
	0	0	0	0
	-288	-360	-480	-720
1.2 $m_{-4} = -\frac{1}{2}$	-300			
2.2 $m_{-3} = -\frac{2}{3}$	-320	-400		
3.2 $m_{-2} = -1$	-360	-480	-600	
4.2 $m_{-1} = 2$	120	240	360	480
	B_0	B_1	B_2	B_3

We get

$$T^{(-4)} = \begin{array}{ccccc} 120 & 240 & 360 & 480 & 600 \\ & -360 & -480 & -600 & -720 \\ & & -320 & -400 & -480 \\ & & & -300 & -360 \\ 0 & & & & -288 \end{array}$$

4. A Proposed Model

We can construct one systolic array that solves the given equations $Tx = B$. Because of the similarities in their operations, processors $P_{\pm k}$ and Q_k $(k \geqslant 0)$ are combined into one super-processor S_k $(k \geqslant 0)$. We then program S_k $(k \geqslant 0)$ to do the regeneration of $T^{(-n)}$ and the solution of $T^{(-n)}x = B^{(-n)}$. Let us describe our linear array of n+1 super-processors $S_0, S_1, \ldots, S_n$. (The last processor S_n is needed for the back substitution). In the Bareiss algorithm four triangular matrices.

$$\alpha = \begin{bmatrix} \alpha_0 & & 0 \\ \alpha_1 & \ddots & \\ \ddots & \alpha_1 & \alpha_0 \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_0 & \beta_1 & \ddots \\ & \ddots & \beta_1 \\ 0 & & \beta_0 \end{bmatrix}$$

$$\gamma = \begin{bmatrix} \gamma_0 & & 0 \\ \gamma_1 & \ddots & \\ \ddots & \gamma_1 & \gamma_0 \end{bmatrix} \qquad \delta = \begin{bmatrix} \delta_0 & \delta_1 & \ddots \\ & \ddots & \delta_1 \\ 0 & & \delta_0 \end{bmatrix}$$

are updated (see Figure 1). Now each processor S_k has registers to store α_k, β_k, γ_k and δ_k. (When describing processor S_k we shall omit the subscripts and simply refer to registers α, β, γ and δ.) Processor S_k requires four additional registers λ_k for a multiplier m_{-j}, μ_k for a multiplier m_{+j}, and ξ_k and η_k which are associated with the right-hand side vector B and the solution x.

Data flows in both directions between adjacent processors, as shown in Figure 5. Hence, each processor needs five input and five output data paths denoted by *inL1, inL2, inR1, inR2, inR3, outL1, outL2, outL3, outR1,* and *outR2* (see Figure 6).

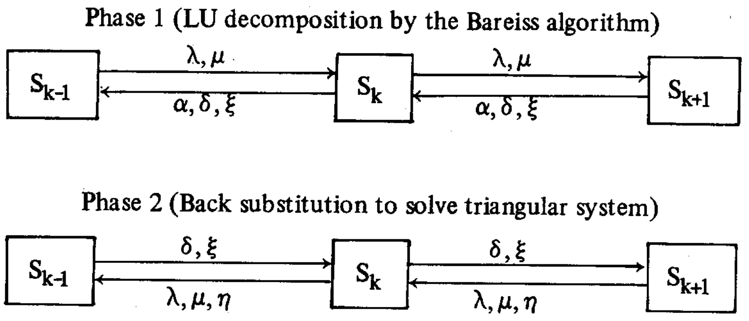

Figure 5. Data flow for systolic system solver.

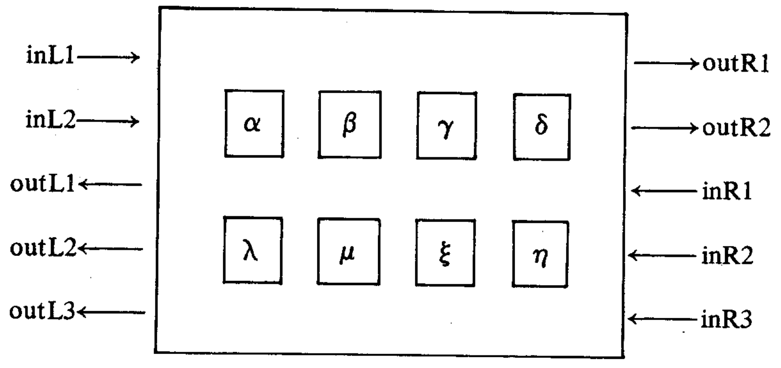

Figure 6. Systolic processor for systems.

Initialization is as follows: $\alpha_k := t_{-(k+1)}$; $\beta_k := t_k$; $\gamma_k := t_{-k}$; $\delta_k := t_{k+1}$; $\lambda_k := 0$; $\mu_k := 0$; $\xi_k := b_{n-k-1}$; $\eta_k := b_{n-k}$; all for $0 \leqslant k \leqslant n$ (we assume that $t_{-(n+1)} = t_{n+1} = b_{-1} = 0$ to cover end-conditions). Clearly this can be done in time order(n) if T and B

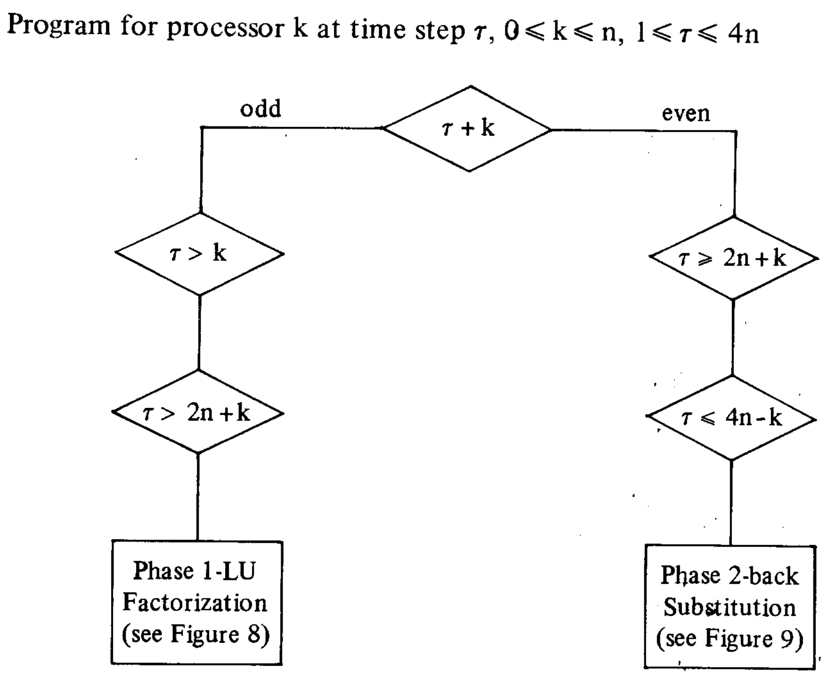
Program for processor k at time step τ, $0 \leqslant k \leqslant n$, $1 \leqslant \tau \leqslant 4n$
odd
$\tau + k$
even
$\tau > k$
$\tau \geqslant 2n+k$
$\tau > 2n+k$
$\tau \leqslant 4n-k$
Phase 1-LU Factorization (see Figure 8)
Phase 2-back Substitution (see Figure 9)

Figure 7. Systolic processor for equation solver.

are available at either end of the systolic array.

We present the program executed by processor S_k $(0 \leqslant k \leqslant n)$ at time step $\tau (1 \leqslant \tau \leqslant 4n)$ in Figure 7. The final solution x is given by $x_k = \xi_k$, where ξ_k is stored in register ξ of processor S_k after step 4n. What follows are some observations concerning the program:

1. Processor S_k is active only if $k < \tau < 2n-k$ (Phase, 1) or $2n+k \leqslant \tau \leqslant 4n-k$ (Phase 2). It is assumed that S_k knows its index k and the current value of τ (though this could be avoided by the use of 1-bit systolic control paths).

2. Pairs of adjacent processors could be combined. Since only one processor of each pair is active at each time step. This would increase the mean processor utilization from 25% to 50% (see observation 1 above).

3. Processor S_0 performs floating-point divisions, other processors perform only additions and multiplications. A time step has to be long enough for six floating-point additions and multiplications, plus data transfers, during Phase 1 (less during Phase 2).

4. The Bareiss algorithm requires $4 \cdot 5n^2$ multiplications as given, but a simple modification (transmitting $1+\lambda\mu$) will give time 4n if have $[\frac{n}{2}]$ processors (see ob-

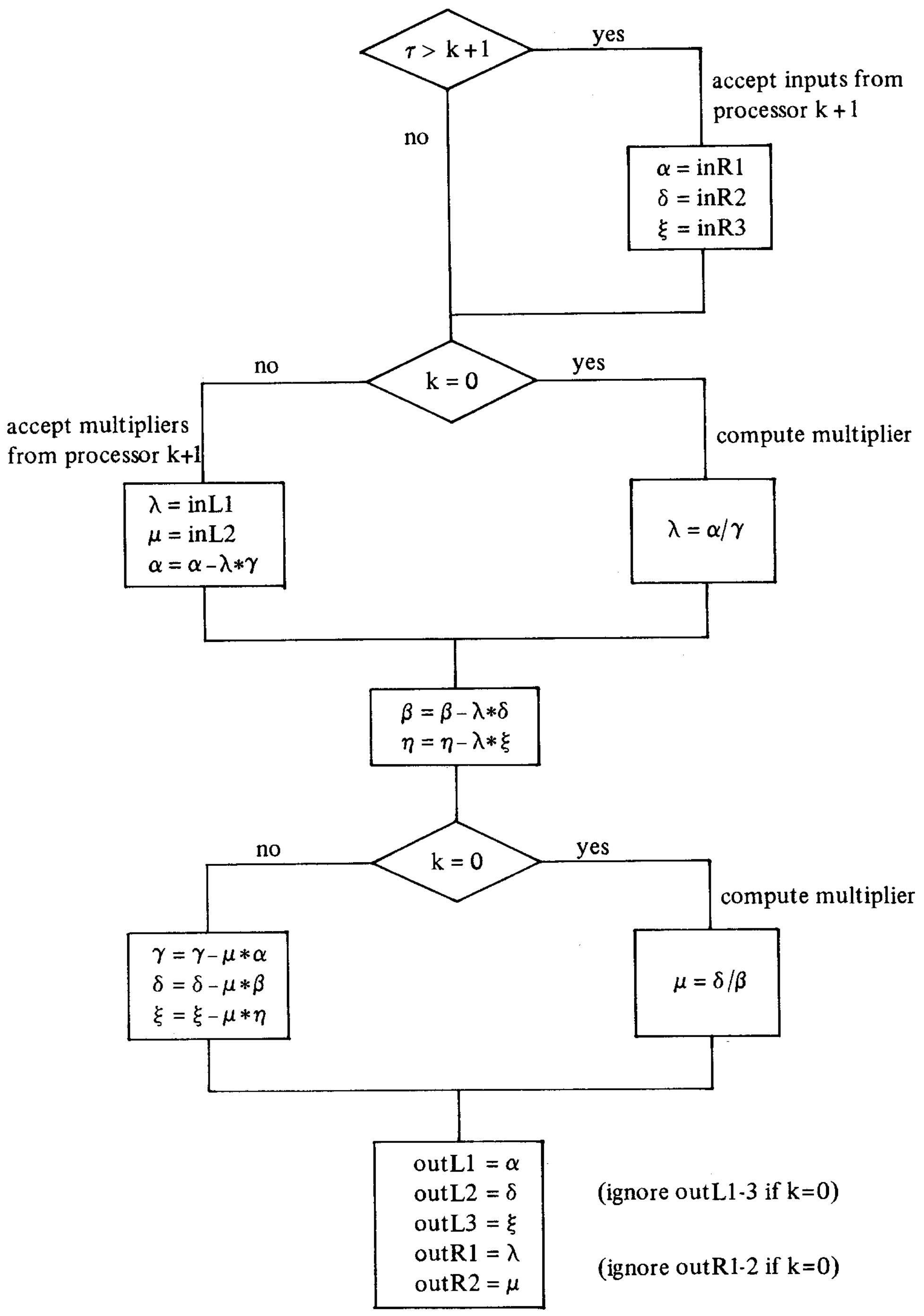
$\tau > k+1$
yes
no
accept inputs from processor k+1
α = inR1
δ = inR2
ξ = inR3
k = 0
no
yes
accept multipliers from processor k+1
λ = inL1
μ = inL2
$\alpha = \alpha - \lambda * \gamma$
compute multiplier
$\lambda = \alpha / \gamma$
$\beta = \beta - \lambda * \delta$
$\eta = \eta - \lambda * \xi$
k = 0
no
yes
$\gamma = \gamma - \mu * \alpha$
$\delta = \delta - \mu * \beta$
$\xi = \xi - \mu * \eta$
compute multiplier
$\mu = \delta / \beta$
outL1 = α
outL2 = δ
outL3 = ξ
outR1 = λ
outR2 = μ
(ignore outL1-3 if k=0)
(ignore outR1-2 if k=0)

Figure 8.

servation 2), each with six multiply-add units. The corresponding figures for the symmetric Bareiss algorithm for symmetric matrices are $4n^2$ 4n, and 5.

5. An alternative for Phase 2 is the use of the Gohberg-semençul formula. But the formula is more expensive in terms of both operations and time, and it also fails to take advantage of the possible band structure of the matrix.

6. Processor S_k typically reads its input lines *inL1*, ... , *inR3*, does some floating-point computations, and writes to its output lines *outL1*, ... , *outR2*. Hence, pairs of input and output lines could be combined into single bidirectional lines (e.g. *inL1* and *outL1* could be combined).

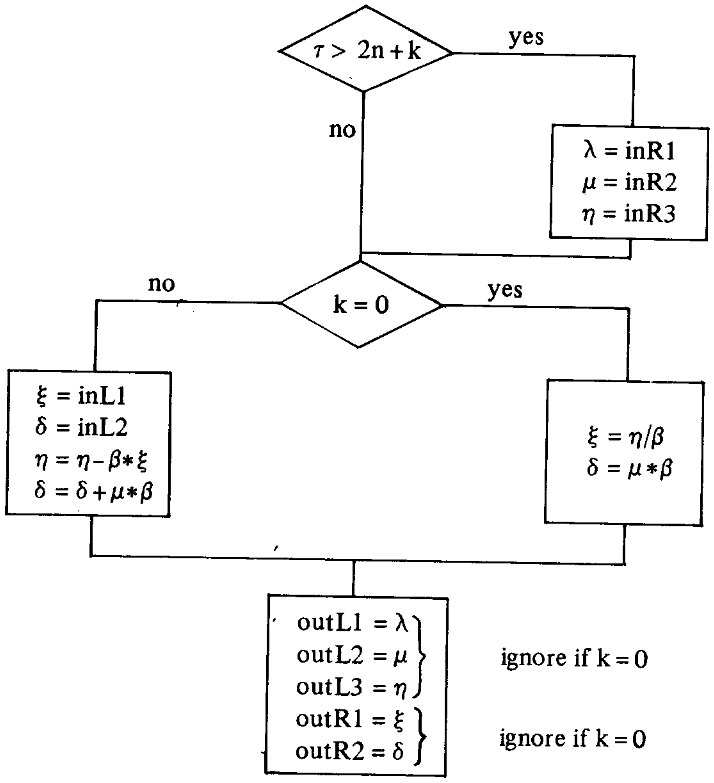

Figure 9.

5. Acknowledgement

The author would like to thank Dr. Fong-Ching Lin, Associate Professor of the Department of Information Engineering of National Taiwan University, for pointing out the references; Dr. Chuang-Ching Song, Associate Professor of the Department of Applied Mathematics of National Chengchi University, for helping me solve some of the mathematical problems during this study and Dr. Dale Bremmer, Associate Professor of the Engineering Technology of Texas A&M, for sending me all the informaticn I requested.

6. References

1. H.T. Kung and C.E. Leiserson, Systolic Arrays (For VLSI) April, 1978, 201-223.
2. H.T. Kung, Let's Design Algorithms for VLSI Systems Jan., 1979, 115-125.
3. E.H. Bareiss, Numberical Solution of Linear Equations with Toeplitz and Vector Toeplitz Matrices, Numer. Math., 13(1969), 404-424.
4. H.T. Kung and C.E. Leiserson, Algorithms for VLSI Processors Arrays in Introduction to VLSI Systems, 1980, 180-208.
5. S.Y. Kung, Impact of VLSI of Modern Signal Processing, Proc, USC Workshop on VLSI and Modern Signal Processing, Los Angeles, Ca (November 1982), 123-132.